KAWASAKI STEEL TECHNICAL REPORT
No.27 (November 1992)
Hot-Rolled, Cold-Rolled and
Surface Coated Steel Sheets
and Electronics and Instrumentation

Development of General-Purpose Internetworking Unit

Tetsuo Ogawa, Norimasa Hattori, Katsumi Minegishi, Takeshi Ueda, Toshiyuki Aikawa

Synopsis :
A general purpose internetworking platform which permits the interconnection of any set of networks has been developed by Kawasaki Steel. The platform consists of several network interfaces with an OS. The OS is designed to realize concurrent processing of protocols by a dynamic task switching architecture based on features of protocol processing. A high-level protocol generating language was also developed, thereby improving protocol development. It is expected that a higher level of network integration can be realized by using this platform in effective, strategic internetworking applications.

The body can be viewed from the next page.

Development of General-Purpose Internetworking Unit*

Tetsuo Ogawa
Staff Manager, Research & Development Dept., Integrated Systems & Electronics Div.

Norimasa Hattori
Staff Assistant Manager, Research & Development Dept., Integrated Systems & Electronics Div.

Katsumi Minegishi
Staff Assistant Manager, Research & Development Dept., Integrated Systems & Electronics Div.

Takeshi Ueda
Research & Development Dept., Integrated Systems & Electronics Div.

Toshiyuki Aikawa
Integrated Systems & Electronics Dept., Kawasaki Steel Systems R&D Corp.

Synopsis:

A general purpose internetworking platform which permits the interconnection of any set of networks has been developed by Kawasaki Steel. The platform consists of several network interfaces with an OS. The OS is designed to realize concurrent processing of protocols by a dynamic task switching architecture based on features of protocol processing. A high-level protocol generating language was also developed, thereby improving protocol development. It is expected that a higher level of network integration can be realized by using this platform in effective, strategic internetworking applications.

1 Introduction

The concept of strategic information system (SIS) has been widely discussed in recent years, and is now becoming a reality. The background of this trend includes changes in social paradigms and technical innovation. To realize a highly information-oriented society, information systems have been actively adopted in various fields, and the level of network construction has shifted from information systems within individual companies to the horizontal development of information networks among companies, and from intra-industry to inter-industry networks. The technical progress represented by the standardization of open systems interconnections (OSI), construction of integrated servides digital networks (ISDNs), and sharing of resources by local area networks (LANs) provides the network infrastructure indispensable to the realization of SISs.[1,2]

In view of these developments, Kawasaki Steel's Systems and Electronics Div. selected "connectivity" and "manageability" as core techniques for development in its network integration business. The term "connectivity" refers to the capability of linking devices and/or networks, which provides the basis for horizontal network expansion. "Manageability" means the capacity of maintaining unified management over the different types of machines, different vendors, and different networks which are integrated for horizontal network expansion. The authors developed a general-purpose internetworking unit as a means of realizing these two technical capabilities.

This report describes the techniques of the internetworking operating system (OS) which supports the general-purpose internetworking unit.

2 Requirements for General-Purpose Internetworking Unit

The information system discussed here is composed of various makes and types of information equipment. The networks which connect this equipment are also diverse. However, the lack of inter-operability among these systems is a major obstacle to progress in the horizontal development of information systems among companies and among industries. Furthermore, although the expansion of networks increases the necessity of integrated management, it is at present very difficult to apply an integrated management architecture; this is also an obstacle to progress in the horizontal development of information systems.

* Originally published in *Kawasaki Steel Giho*, **24**(1992)1, 20–25

On the basis of a concept of "any-to-any" linkage of equipments and networks, which is refered to as internetworking, and management of this internetworking system, the authors began development of a general-purpose internetworking unit.

From the viewpoint of business development, the following two features were required in this unit:

(1) Must contribute to timely commercialization in line with business development.
(2) Must provide the basis for future business expansion.

The following were considered necessary conditions for meeting these requirements:

(1) Must be compatible with various communication interfaces, for example, IEEE (Institute of Electrical and Electronics Engineers) 802.3, V. 24, X. 21, RS-232C, and ISDN, and with those developed in the future.
(2) Must permit loading of various protocols with the communication interfaces in (1), and must be compatible with new protocols.
(3) Must make possible improve the productivity in the development of protocol processing programs.
(4) Must permit loading of application systems such as network management systems.

To satisfy these conditions, the following methods were adopted:

(1) Multiprocessing method using multiprocessor architecture, with a processor board for each communication interface, aimed at the distribution of functions and loads.
(2) Multitasking method for realizing a multi-protocol environment in a single processor, and for executing additional applications such as network management simultaneously with communication.
(3) Loading of an internetworking OS as a means of solving interprocess communication/synchronization, exclusion control, and other problems peculiar to concurrent processing, which are techniques forming the basis of (1) and (2).
(4) High-level protocol generating language for improving the efficiency of protocol processing program development.

3 Configuration of General-Purpose Internetworking Unit

3.1 Hardware Configuration

The general hardware configuration of the general-purpose internetworking unit for realizing the distribution of functions and loads is shown in **Fig. 1**. CPUs are mounted on the main board for control of the entire unit and each of the interface boards which control communication at each interface. These CPUs are linked by a bus. The CPUs on the boards share the system RAM on the main board and exchange information using this RAM. Functions are distributed to each board, and the distribution of loads is made possible by using multiple boards of the same kind.

3.2 Internetworking OS

Figure 2 shows the functional structure of the internetworking OS developed in this work to realize efficient multitasking of protocol tasks.

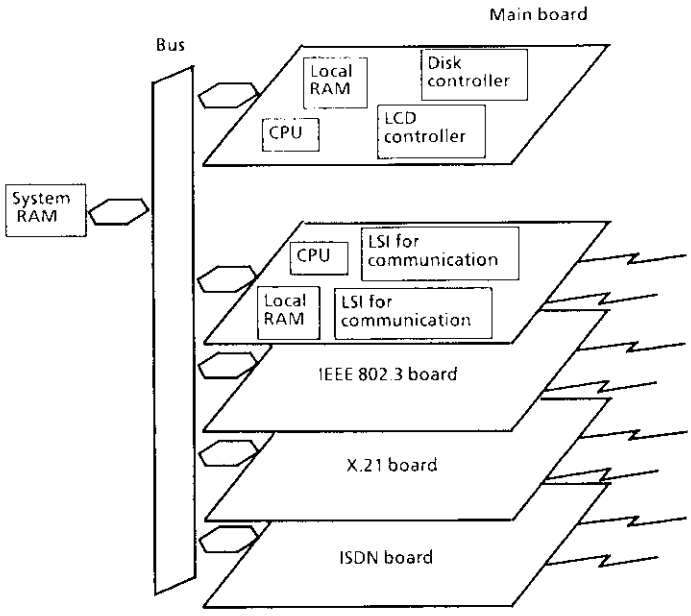

Fig. 1 Hardware block diagram of internetworking unit

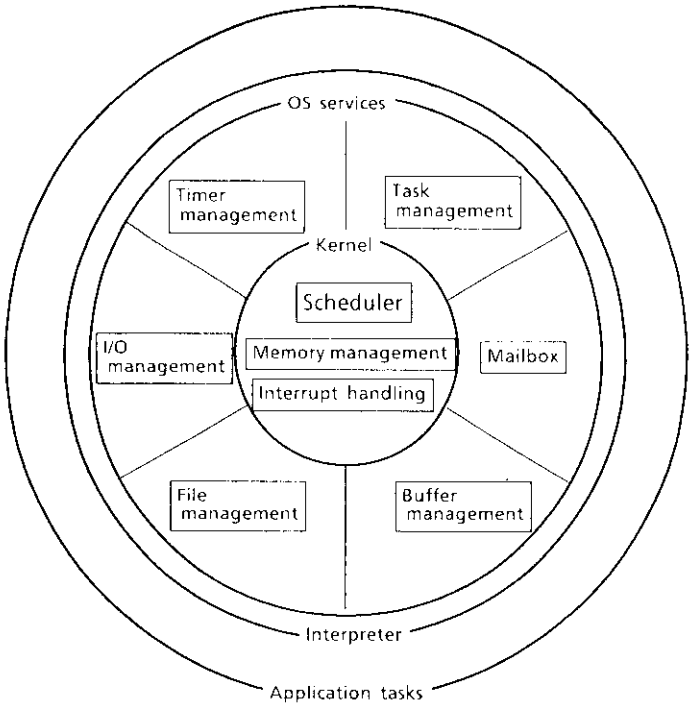

Fig. 2 Internetworking-OS functional structure

A scheduler is provided in the kernel portion to process multiple tasks. OS services for managing resources such as the buffer and timer run around the scheduler. Application tasks run on this OS, but OS services are provides for protocol tasks through a protocol-generating language interpreter.

3.3 Application

The following processing applications can be run on the internetworking OS:
(1) Protocol processing
(2) Device monitoring
(3) Diagnostics
(4) Fault processing
(5) Exchange of routing information
(6) Network management

A high-level protocol generating language was developed to ensure efficient generation of protocol processing programs, which are the main application of this unit.

4 Protocol Processing and Internetworking OS

4.1 Model of Protocol Processing

Figure 3 shows the model of protocol processing, which is mainly executed by the internetworking unit.

The processing steps are outlined below.

(1) Reception Processing

Data is physically received by a communication LSI (external processing) and received data is accepted and forwarded to the forward processing in case of receiving completion interrupt from the communi-

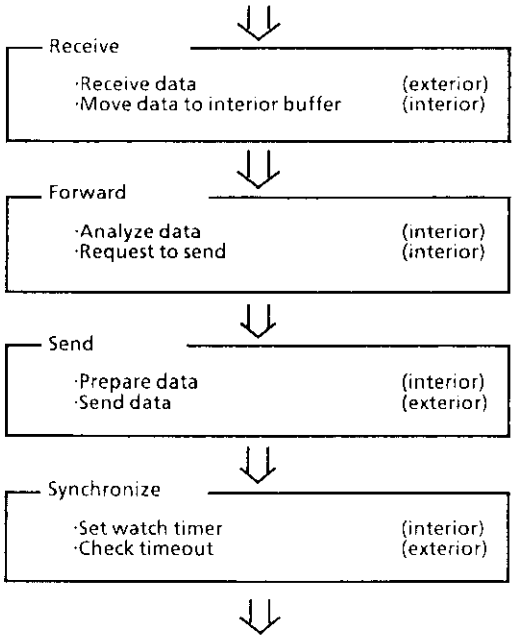

Fig. 3 Protocol process model

cation LSI (internal processing).

(2) Forward Processing
The received data is analyzed and, if it is to be forwarded, a transmission request is made to the destination protocol-processing task (internal processing).

(3) Transmission Processing
Data is prepared for transmission and the communication LSI is instructed to send the data (internal processing); also includes the transmission of the data by the communication LSI (external processing).

(4) Synchronization Processing
This function sets a monitoring timer and then waits for an interruption in the turn either of the receipt of a receiving acknowledgment frame from the sender or expiration of the timer (internal processing), and further includes acknowledgement of the expiration of the timer (external processing).

During each of the processing steps (1) to (4) above, it is necessary to continually observe the condition of the circuit and respond appropriately when major changes occur.

4.2 Characteristics of Protocol Processing

From the model described above, the following two points can be mentioned as characteristics of protocol processing:

(1) The proportion of external processing is high.

(2) A variety of error processing modes are necessary.

External transmission processing by a communication LSI generally requires much more time than processing within the CPU due to its dependence on the external transmission speed. The time to expiration of the synchronization timer is also sufficiently longer than the time required for internal processing. Protocol processing is characterized by a very high proportion of external processing, but because protocol process tasks wait for interruptions during this external processing, the execution of other protocol process tasks or other application tasks is possible with the CPU (**Fig. 4**).

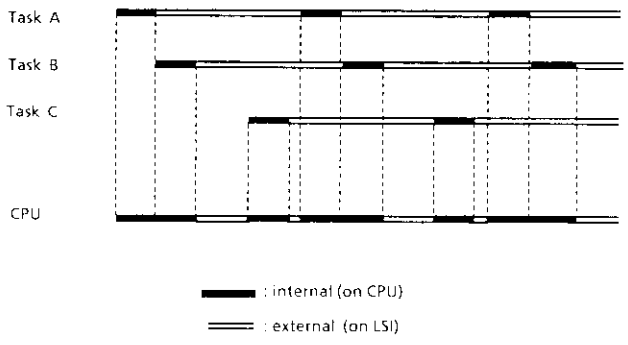

Fig. 4 Multi-tasking of protocol tasks

Because protocol processing normally has external relations with the transmission media and destination end users, an external response to various exceptions is necessary. These exceptions are indicated to the protocol process as errors by interruptions caused by external functions. When these interruptions occur, the protocol process must analyze the cause of error and execute appropriate processing to cope with it. However, error analysis is very complex because the causes of the errors are diverse.

The internetworking OS was developed to ensure efficient concurrent processing of the protocol process task groups with these characteristics. The most distinctive feature of this OS is a high-speed task switching method based on a simple control structure, as described below.

5 Structure of Internetworking OS

5.1 Dynamic Task Switching Method

The task switching method developed in this work to efficiently realize the concurrent processing of protocol process tasks is called here the dynamic task switching method.

In general, a task undergoes a transition of status as shown in **Fig. 5**. In the internetworking OS, a transition to the state of waiting for an event does not occur; rather, the task remains in the ready state while waiting for the occurrence of an event.

Task switching in a conventional multitask OS is basically effected by the static task switching method by a timer, and the task side is not aware of the timing of task switching.[3-6] In contrast, in task switching in this internetworking OS, the task itself dynamically determines the timing of task switching. Concretely, task switching occurs when a task currently being excuted autonomously issues a command to abandon its excu-

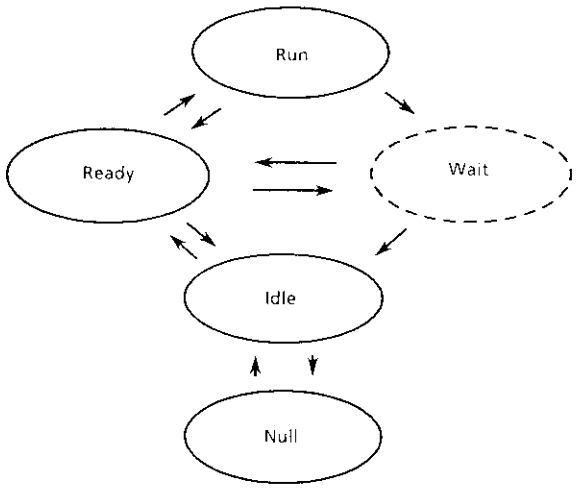

Fig. 5 Transition of task status

tion right. The task which has abandoned the execution right transfers to the ready state and waits to acquire the next execution right. Fundamentally, the execution right is not transferred to another task unless the task being executed abandons its right.

Section 6 describes how efficient concurrent processing of protocol tasks is realized on the basis of this dynamic task switching method.

5.2 Task Status Queue

The only data structure used in realizing the dynamic task switching method is a ready-state queue with a circular list structure.[7] The structure of this task status queue is shown in **Fig. 6**. The queue comprises a series of task control data structure items in the ready state (task control blocks). Among the tasks forming this queue, the task scheduler assigns the execution right to the task indicated by the execution pointer (task B in upper part of Fig. 6). When a command to abandon the execution right is issued by the task being executed, the ready-state queue rotates, the execution pointer indication changes, and the task control block being executed moves to the end of the queue. As a result of the rotation, the task corresponding to the task control block indicated by the execution pointer (task C in lower part of Fig. 6) acquires the execution right. Rotation of the queue is possible with only rewrite processing of the execution pointer; thus, only simple processing is needed in preparation for task switching.

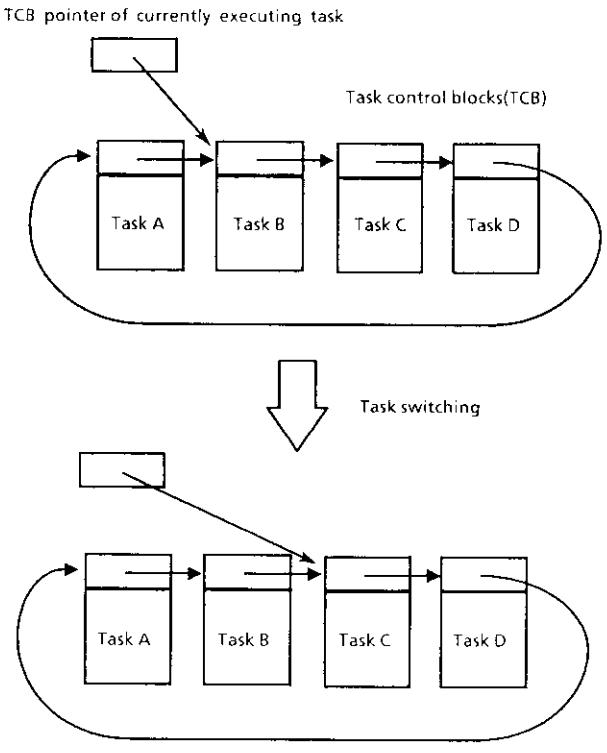

Fig. 6 Snapshot of TCB ready queue linkage in rotation fashion

6 Concurrent Processing of Protocol Process Tasks

6.1 Event-Waiting Operation

In the event-waiting operation with a general-purpose OS, the general practice is that waited events are registered in the OS, and a transition to the event-waiting state occurs.[8,9] When an event occurs, the OS analyzes its cause and switches the task in the event-waiting state to the ready state. The load of the OS is heavy because it executes both the processing for the transition of task status and the check of the occurrence of events.

In the dynamic task switching method, the OS merely gives the execution right to a task in the ready state in the appropriate order and does not distinguish whether a task is waiting for events or not. The waiting task itself checks the occurrence of events and, when events do not occur, instantaneously abandons the execution right.

In protocol processing, the following eventwaiting states occur:

(1) Completion of Transmission (Transmission Processing)
After the protocol process task issues a data transmission command to the communication LSI, it waits for the interruption at the completion of transmission by the LSI.

(2) Data Receiving (Receiving Processing/Synchronization Processing)
The protocol process task issues a data receiving command to the communication LSI in expectation of the arrival of incoming data. In particular, after the interruption at the completion of transmission by the LSI, the protocol process task waits for the receiving of receiving acknowledgment data sent from the sender in order to synchronize communications.

(3) Communication Error
During the execution of external processing, the protocol process task checks whether the continuation of communication has become impossible due to external conditions.

In (1) and (2) above, the monitoring timer is started, and waiting covers both the interruption from the LSI and that from the timer. In both cases, the event-waiting state occurs immediately after the command from the protocol process task side. In contrast, in case (3), continual monitoring is necessary because external conditions such as transmission media change constantly; in this case, the event-waiting state can be considered continuous.

In the dynamic task switching method, efficient concurrent processing is possible with tasks having the following two differing types of event-waiting states:

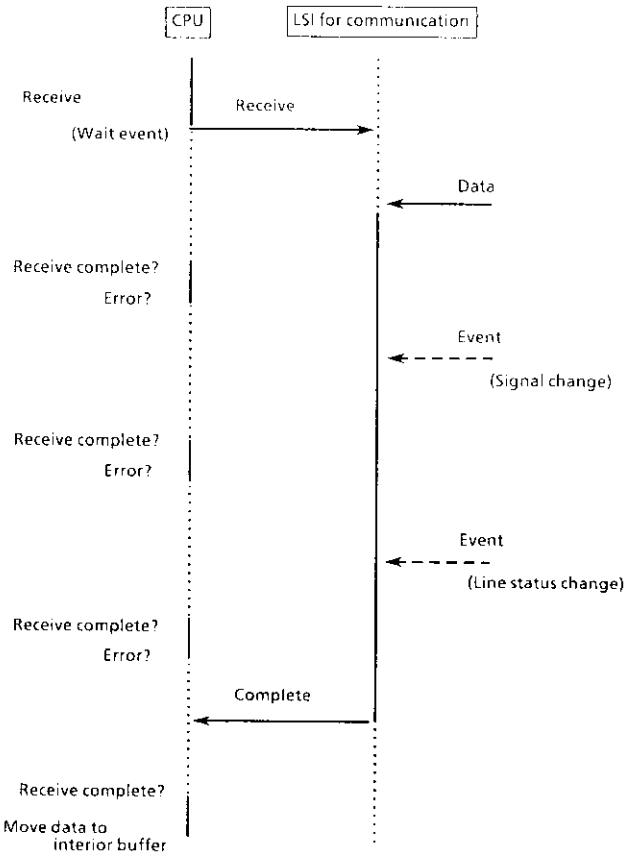

Fig. 7 Event wait mechanism between CPU and external events under "receive" semantics

(1) Discretely occurring and predictable event-waiting states
(2) Continuous event-waiting states

The concept for the realization of this is shown in **Fig. 7**.

6.2 Switching Overhead

Task switching involves the following two types of overhead:

(1) Overhead for state transition from the execution state
(2) Overhead for event processing

These two types of overhead are discussed in the following.

6.2.1 Overhead for state transition from execution state

When a transition from the execution state to the event-waiting state occurs, the waited event is registered. When many types of waited events exist, as with protocol process tasks, overhead is relatively large due to the large volume of registery processing required.

When a transition from the execution state to the ready state occurs, however, this type of processing is unnecessary, and the OS merely removes the execution right from the task. Therefore, the overhead for state transition in waiting for events in the dynamic task switching method is smaller than that during event waiting with a conventional OS.

6.2.2 Overhead for event processing

With the conventional (static switching) method, tasks are switched to the event-waiting state when an incoming event is registered. The OS checks the occurrence of events, but until an event occurs, the event-waiting task is not in the ready state and cannot obtain the execution right. In the dynamic task switching method, however, because the task checks the occurrence of events, the execution right is also transferred to an event-waiting task. It might seem that the efficiency of concurrent task processing is higher when the conventional method of transition to the event-waiting state is applied, because there is a corresponding decrease in the number of tasks in the ready state.

However, when the check of event occurrence and the subsequent initiation of processing are considered, it can be seen that a considerable load is placed on the OS, which must perform all this processing. It is also possible that multiple tasks are waiting for one event, and that multiple processing operation should be initiated. Event processing is very complex in an environment in which there is a strong possibility that multiple tasks are waiting simultaneously for multiple events.

In contrast, in the dynamic task switching environment, the task itself checks the occurrence of events. Because the task side determines the processing to be initiated after an event occurs and the classes of events to be checked, processing can be performed with relatively high efficiency after events occur. The OS, provided it has the means of informing all tasks of the occurrence of events, need only assign the execution right to tasks, and this processing is also simple. Therefore, processing overhead can be held to a minimum.

6.3 Assurance of Real-Time Characteristics

Because the timing of task switching is determined by the task side in the dynamic task switching method, the possibility remains that the OS may become uncontrollable. In other words, if a task refuses to abandon the execution right, the system becomes inoperable. As a remedy for this, this OS is equipped with a static task switching function, which operates at relatively long intervals, and a means of allowing the user to order the abandonment of the execution right.

When an application task with a high proportion of internal processing is mounted, the timing of task switching should be taken into consideration during the generation of an application program. It is recommended that the timing of task switching be selected within a range which ensures the real-time characteristics of protocol processing when a command for the abandonment of the execution right is used.

However, because the real-time characteristics of pro-

tocol tasks are assured by static task switching even in the worst case, the operation of the system can meet the requirements of the protocol. To ensure more efficient concurrent processing, it may be necessary to make more effective use of commands for the abandonment of the execution right.

7 High-Level Protocol Generating Language

A high-level protocol generating language is a language developed so that the programmer need not be aware of the complexity of the protocol processing or the concurrent processing of protocol process tasks when performing programming in the development of protocol processing programs. The programmer can generate protocol processing programs merely by arranging functions according to the flow of protocol processing.

The execution of dynamic task switching for efficient concurrent processing of protocol tasks, as mentioned above, is also hidden in the semantics generated by a high-level language interpreter. In other words, in the semantics of a high-level protocol generating language, a command for the abandonment of the execution right is implicitly included within a command for external operation waiting events, and task switching processing is initiated automatically. As a result, the generated protocol processing program provides high efficiency concurrent processing even if the programmer is not conscious of efficiency.

This high-level protocol generating language makes possible the expandability of the general-purpose internetworking system and improvement in the efficiency of protocol generation.

8 Conclusions

Kawasaki Steel's Systems and Electronics Div. developed hardware, an internetworking OS, and a high-level protocol generating language as the platform of a general-purpose internetworking unit as a powerful tool for future network integration business.

This unit is intended to provide the interoperability of any set of networks from the view of connectivity and manageability. The unit itself is composed of communication interface boards suitable for various types of networks and a main board for the overall control of the unit. Under the internetworking OS of the CPUs mounted on the boards, various kinds of protocol program and other application programs run as tasks.

The following results were obtained in the development of this unit:

(1) By adopting the dynamic task switching method, processing efficiency was improved in the concurrent processing of protocol process tasks with a high proportion of external processing and various classes of waited events.

(2) It has become possible for protocol processing program development personnel to develop programs with a high concurrent processing efficiency in a short time without being conscious of the details or concurrency of protocol processing.

The authors plan to mount various protocols and protocol conversion functions on this unit in the future, using the unit as a tool for flexibly meeting customer needs. Furthermore, a higher level of network integration will be possible by mounting applications such as a network management function on this unit.

References

1) H. Ishida, T. Tokuda and H. Tokuda: "Computer Network", (1986), [Kyoritsu-Shuppan]
2) Y. Matsushita: "Computer Network", (1983), [Baifukan]
3) A. S. Tanenbaum: "Operating Systems: Design and Inplementation", (1987), [Prentice-Hall]
4) M. J. Bach: "The Design of the UNIX Operating System", (1986), [Prentice-Hall]
5) P. J. Fortier: "The Design of Distributed Operating System", (1988), [Addison-Wesley]
6) L. Bic and A. C. Shaw: "The Logical Design of Operating Systems", (1988), [Prentice-Hall]
7) A. V. Aho, J. E. Hopcroft and H. D. Ullman: "Data Structure and Algorithm", (1983), [Addison-Wesley]
8) NEC Corp.: "RX616 kernel user's manual", (1989)
9) OTC Corp.: "pSOS+ Series Software Summary", (1988)